CMPSC265 Data Structures and Algorithms
Homework 11

Due: Dec 11th, 2019, 11:59pm

Learning Objectives

- Graph
- Minimum Spanning Tree
- Shortest paths
- Kruskal algorithm
- Bellman_Ford algorithm
- Floyd-Warshall algorithm

Deliverables:

- The modified java file Graph.java for Problem 1, Problem 2 and Problem 3.

Instructions:

- Please first download the Java source file *UnionFind.java* that defines a class for Union_Find. The class supports the operations as detecting whether two nodes belong to the same union, generating a union of two sub-unions, and etc.
- Please also download *Graph.java*, modify the codes based upon requirements, and submit the updated source file. Please do NOT change the provided part of the codes, but you may add additional methods if needed.
- Please add the required *credit comments* and *comments* to each Java program you submit.
- **Please TRY YOUR BEST before seeking help from any online resources or tutors or other people. If you have asked for any kind of help, you need also submit your own draft before getting the help, and write a report of how you got helped and what you have learned from the help.**

Problem Specifications:

Problem 1: Finding Minimum Spanning Tree using Kruskal Algorithm (35')

The attached Graph.java file contains the definition for the Vertex Class, the Edge class as well as for the Graph class.

Please finish the implementation of the method of *kruskalMST()* which will find the minimum spanning tree on a connected undirected weighted graph.

The Kruskal's algorithm for finding minimum spanning tree is as follows:

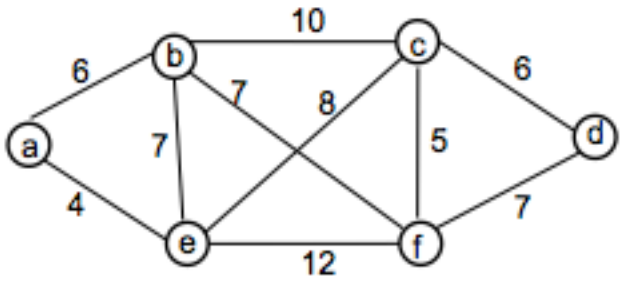
- Initially, the MST contains all vertices but no edges.
- Sort the edges in a non-decreasing order.
- Go over (sorted list of) edges and add an edge to the MST. Important: If the edge makes a cycle, ignore it.
- Continue until N-1 edges are added, when MST is formed.

How to detect whether adding an edge can make a cycle? We can use a *union_find* data structure whose implementation has been provided to you. It supports operations as makeSet(), find(), union().

- First, we can make N set (one for each vertex on the graph)
- Every time we want to add an edge *ab* to the spanning tree, we first check whether find(a)==find(b), indicating whether they belong to the same union. If true, it will make a cycle, and we cannot add edge *ab*. Otherwise, the edge *ab* will be added to MST, and we union the two sub-unions where a and b belongs to by union(a, b).

<u>Outputs:</u>

Given the following graph:



```
$ java Graph
Minimum spanning tree:
a----e: 4
c----f: 5
c----d: 6
a----b: 6
f----b: 7
```

Problem 2: ***Finding single-source shortest paths using Bellman-Ford algorithm (35')***

Description:

Please finish the implementation of the method of *bellman_ford(source)* which finds and returns the shortest paths as well as the weights of the shortest paths from the source vertex to all other vertices on a connected undirected weighted graph.

Bellman-Ford algorithm adopts dynamic programming, which uses a table (one-dimensional array of size N) to save distance to each node, initialize with infinity except for the source which is 0 distance from itself. It is an iterative solution, and needs to iterate N-1 (N here is the number of vertices in graph) times, in each iteration, it will look at all edges and see if you need to update distance from source (update table). For edge u->v do: D(v) = min[D(v), D(u) + w(u,v)].

Here follows is the pseudocodes for Bellman-Ford algorithm:

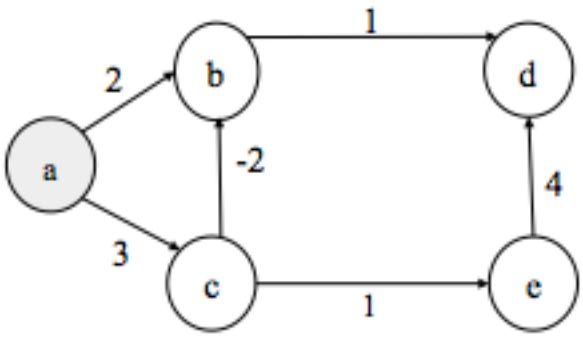
Pseudo-code for Bellman-Ford:

```
Initialize the tables/vectors d and parent
for i := 1 to N - 1 do
    for each edge (u, v) ∈ E do
        if d[v] > d[u] + w(u, v)
            then  d[v] := d[u] + w(u, v)
                  parent[v] := u

for each edge (u, v) ∈ E do
        if d[v] > d[u] + w(u, v)
            then return false   // there is a negative cycle
return true
```

Outputs:

Given the following graph:




```
$java Graph
          Shortest Path     Weights of the Shortest Path
a-b       a->c->b           1
a-c       a->c              3
a-d       a->c->b->d        2
a-e       a->c->e           4
```

Problem 3: Finding all pairs shortest paths using Floyd_Warshall algorithm (30')

Description:

Floyd_Warshall is an algorithm for finding all pairs shortest paths in a weighted graph with positive or negative edge weights (but with no

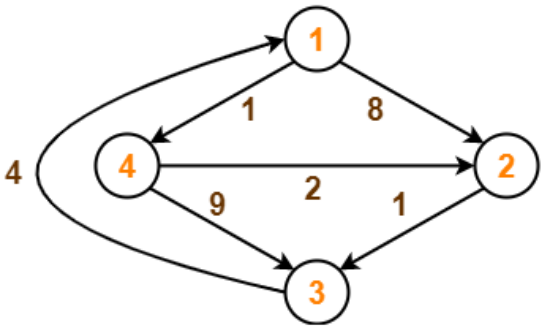
negative cycles). Its basic idea is to one by one pick all vertices and updates all shortest paths that include the picked vertex as an intermediate vertex in the shortest path.

Here follows show the pseudocode of Floyd_Warshall algorithm.

```
for (int k = 1; k =< N; k++)
   for (int i = 1; i =< N; i++)
      for (int j = 1; j =< N; j++)
         if ( ( d[i][k]+ d[k][j] ) < d[i][j] )
             d[i][j] = d[i][k]+ d[k][j];
```

Outputs:

Given the following graph:



$java Graph

The shortest paths between all pairs is:

	1	2	3	4
1	0	3	4	1
2	5	0	1	6
3	4	7	0	5
4	7	2	3	0